LA-UR-10-02112

Approved for public release;
distribution is unlimited.

Title: Interactive Remote Large-Scale Data Visualization via Prioritized Multi-resolution Streaming

Author(s):
James P. Ahrens 113788 CCS-1
Woodring Jonathan L. 209118 CCS-1
Patchett John M. 148176 CCS-1
Maltrud Mathew E. 094509 T-3
DeMarle David E. Kitware Inc.

Intended for: Workshop on Ultrascale Visualization, Nov. 2009

Interactive Remote Large-Scale Data Visualization via Prioritized Multi-resolution Streaming

James P. Ahrens
Los Alamos National Laboratory
ahrens@lanl.gov

Jonathan Woodring
Los Alamos National Laboratory
woodring@lanl.gov

David E. DeMarle
Kitware Inc.
dave.demarle@kitware.com

John Patchett
Los Alamos National Laboratory
patchett@lanl.gov

Mathew Maltrud
Los Alamos National Laboratory
maltrud@lanl.gov

ABSTRACT

The simulations that run on petascale and future exascale supercomputers pose a difficult challenge for scientists to visualize and analyze their results remotely. They are limited in their ability to interactively visualize their data mainly due to limited network bandwidth associated with sending and reading large data at a distance. To tackle this issue, we provide a generalized distance visualization architecture for large remote data that aims to provide interactive analysis. We achieve this through a prioritized, multi-resolution, streaming architecture. Since the original data size is several orders of magnitude greater than the display and network technologies, we stream downsampled versions of representation data over time to complete a visualization using fast local rendering. This technique provides the necessary interactivity and full-resolution results dynamically on demand while maintaining a full-featured visualization framework.

Categories and Subject Descriptors

I.3.2 [**Computer Graphics**]: Graphics Systems—*remote systems*

General Terms

Performance

Keywords

data intensive supercomputing, large scale data, distance visualization, remote visualization, visualization systems

1. INTRODUCTION

For Department of Energy (DOE) application teams, visualizing, analyzing, and understanding their results is the key to effective science. These activities are significantly hampered by the fact that the scientists and the supercomputing resources they work on are located in geographically different locations. The most significant barrier to effective distance visualization is the lack of network bandwidth between sites. The Office of Science (SC) report entitled, "DOE Science Networking Challenge" documents that SC applications are generating massive data at unprecedented rates [15]. Networks are not keeping pace with scale of data being produced, and the report makes a plea for additional networking resources to keep up with the flood of data. It is important to note, that given the mismatch in speed between network technology and the rate supercomputers can generate data, a user can still be "remote" to their local supercomputer due to network speeds. Improving the networking architecture is critical, however, this research offers to reduce the demand on the network by providing a smarter visualization and analysis system for current and future networks.

As part of the analysis process, there is data transferred from a remote site to a local scientist, which can be the results of a visualization or the scientific data set itself. A rough estimate of the amount of data that can be transferred during one day over an optimistic gigabit network is shown in the Table 1. For clarity, Table 2 highlights the disparity between display and wide-area networking technology, i.e., mega/gigapixel displays and mega/gigabit networks, to our supercomputers, which can generate data in the tera-, peta-, and in the future, exascale ranges. Even with increasing network and display technology, we can expect that supercomputing will be at the leading edge of technology, generating data sizes several magnitudes greater than the corresponding network speeds. Even with terascale data sizes it is a significant challenge to transfer, store and interactively visualize data of this magnitude. The transferring of data at this magnitude, as Table 1 shows, would take hours and up to days to move.

Amount per day	108,000,000 MB	108,000 GB	108 TB	0.108 PB

Table 1: An approximation of the amount of data that can be transferred over an optimistic 10 Gbit network.

UVW '09 November 16, 2009 Portland, Oregon

Prefix	Mega	Giga	Tera	Peta
Data sizes	10^6	10^9	10^{12}	10^{15}
Technology trends	Displays and Networks	→	Data Sizes	→

Table 2: The prefixes for units of measure with corresponding technology trends, showing the disparity between local displays, wide-area networks, and supercomputing data sizes.

One approach to address the remote visualization problem is to visualize scientific data at the supercomputing site and send the resulting images over the network. In an *image-based* approach, seen in Figure 1, all of the analysis and rendering is done at the supercomputing site, which has the advantage of sending fixed data size images continuously, consuming a constant bandwidth. A disadvantage to the image-based approach is that is it inherently network bandwidth and latency limited, as frames can only come as fast as the network allows. While image compression and other methods can be used to reduce the network bandwidth utilization, the network latency penalty is always going factor in, reducing the effective frame rate and interaction speed. Another approach is to transfer representations, calculated on demand by the supercomputer, and utilize local rendering. The *representation-based* approach, also seen in Figure 1, has the disadvantage that the latency to the first image can be slower than an image-based approach, as the data can be much larger than a single image as shown in Table 2. Also, the representation can be larger than the client side memory, assuming that local machines are not the same scale as the remote supercomputer.

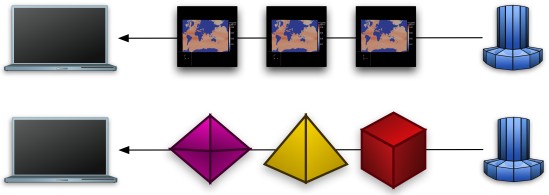

Figure 1: The conceptual difference between an image-based distance visualization (top) and representation-based distance visualization (bottom). Image-based sends images over time, while representation-based sends processed data over time.

The missed opportunity of the image-based approach is that it is an all or nothing proposition, where it assumes there is little processing power at the scientist's local site. It always depends on remote supercomputing resources for rendering, and the frame rate and interaction latency during visualization will be network bound. A scientist's local computing resources can have a significant amount of memory and computational power to perform fast local rendering of selected subsets of data. Our contribution is a description of an interactive remote large-scale data visualization framework that supports prioritized, multi-resolution data streaming.

In a multi-resolution representation based-approach, we send chunks of data from the supercomputer by processing a data set in a multi-resolution manner and sending appropriate data at various resolutions. If we send data chunks that match the network speed, which also matches the display size, as seen in Table 2, this provides the capability of interactivity at display resolution. Immediate visualization is achieved through low resolutions, and full resolution visualization is provided over time by streaming high-resolution representations.

We use local rendering to offload the remote supercomputer and network bandwidth during visual interaction, such as rotation, panning, and zooming. This allows the supercomputer to continuously stream in prioritized high-resolution data during interaction, particularly in prioritized focus areas, such as zoomed in areas. Client side rendering achieves high frame rates and low interaction latency, compared to sending images, while allowing the supercomputer to continuously filter data.

Multi-resolution also lowers the initial latency to the first frame in this approach by the mere virtue of sending small network sized data chunks through prioritized multi-resolution visualization. This approach also ensures that we do not exceed the memory limitations of the local computers that are performing the rendering. We also record meta-data used for data prioritization and culling through multi-resolution sampling of a data set. Multi-resolution data also reduces the I/O time spent by the remote supercomputer and reduces overall processing of low resolution data.

2. PREVIOUS AND RELATED WORK

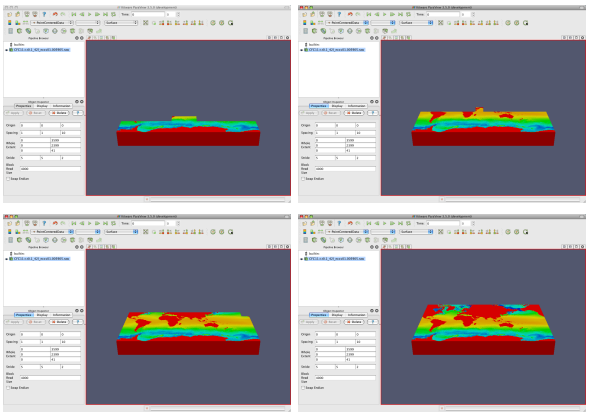

Figure 2: Previous work on large scale visualization in VTK showing a sequence of prioritized streaming of a surface coloring of the CFC concentration in simulated ocean data. The sequence proceeds from left to right, top to bottom. The ocean data is divided into hundreds of file blocks and progressively streamed to the local rendering resource. The blocks fill in from front to back. This is because the pieces are prioritized by shortest distance from the camera. This system is also abstractly represented by Figure 5.

Massive data set sizes can make visualization difficult or impossible, we previously designed a visualization architec-

ture based on VTK [17] that would reorganize a large data set into smaller pieces, and operationally stream the pieces incrementally to generate a visualization [1]. This architecture reduced I/O and processing time by only operating on pieces that contribute to the final result. This was achieved through prioritization and culling of data blocks by meta-data characteristics. The processing order of pieces is such that pieces are displayed incrementally based on estimating the importance of their contribution to the final image through meta-data. Each visualization operation in the processing pipeline, such as reading, clipping, isosurfacing, and rendering, can independently generate a partial priority estimate based on view and data characteristics to generate a final combined estimate to cull and order pieces. Using the new architecture, a visualization renders in a significantly shorter time than the time it takes the standard VTK architecture to create a final image, along with low latency to the appearance of important portions via data prioritization. Figure 2 shows a sequence of streaming ocean data with the streaming system architecture available in ParaView [2, 10].

Our current work is built upon this streaming premise to be able to handle large data, but modifies the architecture to use multi-resolution data to match display sizes and network bandwidth for interactive distance visualization. The past system had difficulty in handling the case for visualizing an entire data set, as the system would have to touch and stream the entire data at full resolution, while the new system provides a much faster multi-resolution solution for immediate results.

Other distance and large scale visualization systems that focus on the delivery and prioritization. The remote visualization found in VTK and ParaView [3], as well as VisIt [4] and Ensight Gold, employ the client/render server/data server architecture to transmit images and/or geometry for large data visualization. Luke and Hansen study the spectrum of visualization system schemes for distance visualization that use the client, server, or both [11]. Klosowski and Silva incorporate priority for a rendering budget system that attempts to prioritize rendering for best-effort results in a limited time frame [8]. Corrêa et al. [7] uses scene visibility for a pre-fetching prioritization in interactive out-of-core rendering of large models.

Visualization systems can handle large data by using multi-resolution techniques, where coarse resolution is used in overviews and fine-resolution is used in details for focus+context visualization. Stolte et al. describe a formalism and different methods of zooming for multi-resolution visualization [18].

Our system is built upon multi-resolution visualization of structured field data, so we compare ourselves to similar multi-resolution visualization methods. To produce a piece of a requested resolution in our system, a read operation reads a spatial block of sampled multi-resolution data with meta-data prioritization information. LaMar et al. [9], Norton and Rockwood [12], Clyne and Rast [6], and Wang et al. [19] use multi-resolution wavelet decomposition and reconstruction to store data at various levels-of-detail for rendering. Childs et al. use a kernel-based resampling method to reduce the footprint of a large data set to scale down the data to fit in memory [5]. Pascucci and Frank use a data indexing scheme using a progressive z-curve reordering and indexing for multi-resolution traversals and visualization of large regular grids [13]. Prohaska et al. present a multi-resolution renderer that uses the HDF5 file format and reader to store blocks and writes preview low-resolution data to disk as needed [14]. Rusinkiewicz and Levoy [16] use a streaming splatting system which progressively refines images while being streamed over the network.

3. METHODOLOGY OVERVIEW

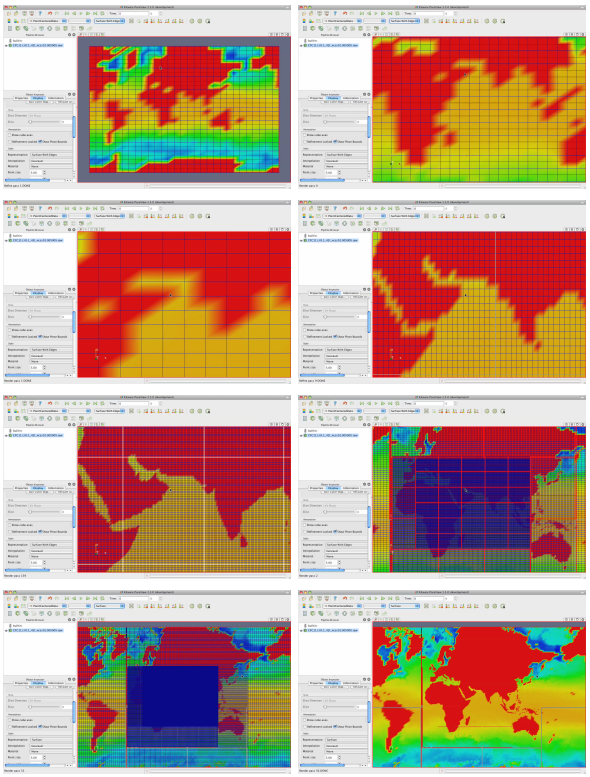

Figure 3: Our streaming multi-resolution visualization system architecture applied to climate data. The visualization progresses from top to bottom, left to right. Data is read from disk using a multi-resolution representation of the data to reduce the amount of data sent over the network which is rendered client side for fast interaction. The visualization starts at a coarse resolution for a fast global overview. In a zoomed in portion, the data is refined and streamed to the client which progressively updates the visualization. The rest of the data outside the view frustum is not touched and left at coarse resolution. The blue lines show grid granularity differences between coarse and fine grain regions. This system is also abstractly represented in Figure 6.

Our focus is a prioritized, multi-resolution representation-based remote visualization approach. Based on our previous observations in the introduction, we will rely on the supercomputer that stores user data as part of the visualization system for reading, computing, and sending prioritized subsets of multi-resolution data. The client side will perform rendering after the supercomputer has sent the appropriate level-of-detail data for fast visual interaction. We prioritize multi-resolution data transfers by sending important low-resolution pieces of data over the network first, and progres-

sively refining them. Once the data is local, scientists can do visualization interactively, offloading interaction tasks from supercomputing resources.

The key additions to our remote visualization system are:

- a server side (supercomputer) data reader that is able to provide multi-resolution pieces of data
- a client side (local computer) multi-resolution renderer
- visualization pipeline meta-information that allows for the reader, renderer, and in-between filters to be able to communicate resolution information

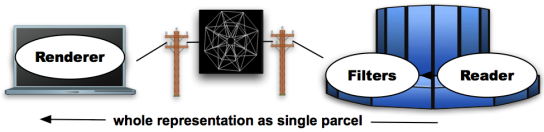

Figure 4: The original VTK/ParaView pipeline.

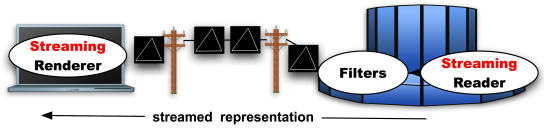

Figure 5: The VTK/ParaView pipeline with prioritized streaming support.

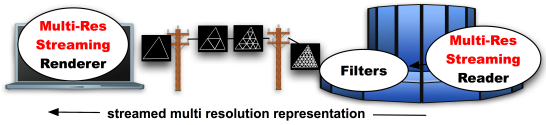

Figure 6: The VTK/ParaView pipeline with prioritized multi-resolution streaming support.

Figure 4 shows the original visualization pipeline. Figure 5 shows the streaming visualization architecture, also shown in Figure 2. Figure 6 shows our new multi-resolution distance model, mirrored by Figure 3 showing the actual system in operation which has been implemented in VTK/ParaView.

To prioritize data transfers, we leveraged our streaming techniques [1] for processing massive data. Our previous solution is to divide a large data set into smaller pieces and then to stream the pieces incrementally through memory, running the visualization algorithms on each piece. A key aspect of this out-of-core/streaming approach is that once the data is divided into pieces, decisions can be made about which pieces are processed and in what order.

Our addition to the culling and prioritization architecture is support for pieces of varying resolutions from structured grids. The resolution of a piece is improved over time based on its priority. This allows the architecture to make trade-offs between improved performance through sending low-resolution data, to match the network bottleneck, with increasing resolution over time. By showing data at full-resolution in areas of interest, and coarser resolution in other areas, the scientist is able to focus in detailed areas of interest quickly, through optimized utilization of the network.

Increasing resolution is done through a spatial split, where a low-resolution piece is spatially split into a number of children with higher resolution, but with the same data size for network streaming and memory footprint limitations. A priority value is calculated for each piece, where high priority pieces will eventually be replaced by higher resolution children in time. In a piece replacement or refinement, a piece's resolution is improved by reading in improved resolution children from disk, sending it up the pipeline for filtering, which eventually reaches the client, who incrementally updates the visualization.

The system architecture via the multi-resolution renderer drives the piece splitting and refining process, by requesting the children to replace a low-resolution parent, up the pipeline chain. It chooses the piece with the highest priority from a priority queue to refine. The multi-resolution reader sends the children down the pipeline and the child pieces are sorted back into the priority queue for further future refinement to higher resolution. With this architecture, the new system creates a streaming multi-resolution distance visualization that increases in resolution over time. This multi-resolution functionality is general enough to work with all visualization operations, such as isosurfaces, cut planes, and clipping, to produce multi-resolution streaming results.

4. IMPLEMENTATION

We utilize the VTK/ParaView streaming, prioritized, culling architecture [1, 17], as the starting point for our distance visualization. This architecture is used for a distance visualization in a representation-based (data or geometry transfer) mode. The reader and filters reside on the supercomputer, while the renderer runs on the client side, connected by a wide area network. The renderer requests data pieces to fill in a visualization, which are incrementally served in priority order by the supercomputer running the reader and filters on the data set. We add multi-resolution visualization to this pipeline, as the original streaming architecture runs at full data resolution. Streaming at full data resolution (Figure 5) has the drawback of taking a very long time for overview visualizations, and our multi-resolution solution (Figure 6) aims to improve it.

4.1 Multi-resolution Reader

In order to perform multi-resolution visualization, many systems, noted in the related work, preprocess a data set in place to generate a multi-resolution or progressive format for fast reading of level-of-detail data. Preserving the data as it is stored is a fundamental requirement for many of our application teams. It can be very time and space consuming to rewrite the data into a secondary format due to nearly random seeks. In many simulation codes, writing the results usually only happens intermittently for select time steps, because it takes so much time, compared to computation, to write out the data. Also, there is a general reluctance by the scientists to rewrite their data in place, due to the extra time and effort and wariness that their data may be corrupted or lost.

Our alternative for accessing data is a multi-resolution data reader that preserves the original data at the highest resolution and creates additional files to store multi-resolution data for coarser resolutions. The multi-resolution

representations form a virtual tree structure with a specific height and degree. The leaves of the tree are at the full resolution of the data, which is stored/left unaltered in the original data file. Each level up the tree is a multi-resolution data sampling of the higher resolution data lower in the tree. The tree is created in the following way:

- The user selects a *degree* where 2^{degree} is the number of children of each parent node in the tree.
- The user selects a *stride* that will stride through the data. The sampling at a given level k is defined by $stride^{degree*k}$.
- The user selects a *height*, which is the number of levels of resolution to create.
- The system reads the user's data file of size D bytes and writes the $height - 1$ resolution levels to disk.

The system creates a multi-resolution representation of the data by sampling the data, and writes the multi-resolution levels and meta-data to disk as independent files while keeping the full-resolution file intact.

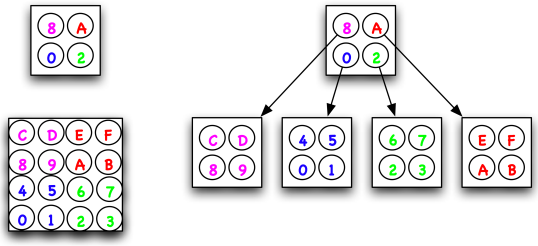

Figure 7: A multi-resolution representation tree with $stride = 2$, $degree = 2$ and $height = 2$. The box in the upper left represents the multi-resolution file for level 0 and the box on the lower left is the original file and the full data for level 1. The tree on the right is the symbolic view of the multi-resolution representation of the data, which is a virtual spatial organization of the data. As can be seen on the left, data and multi-resolution data is actually stored in the original file format/order, but at various strides (samplings).

The purpose of the additional multi-resolution files is to significantly speed up read access to the coarser resolutions. Attempting to read this data from the original file by striding (adaptively sampling) is too time consuming due to the strides and seeks needed to obtain data from disk. By writing additional files, organized for fast read access on low resolution, we can significantly reduce the read access for multi-resolution data. The original data represents the fully refined, highest resolution level and is not modified at all.

The computational cost of this generation step is reasonable, roughly $O(D)$ in time because the entire file is read at a cost of $O(D)$ and multi-resolution representations are written to disk at a cost of $O(D)$. Specifically, the original file is read an element at a time and the element is written to all multi-resolution files that it resides in. The size of the additional files containing multi-resolution representations can be calculated as sum of a geometric series. This formula is given as:

$$\sum_{k=1}^{height-1} r^k = \frac{r^{height} - r}{r - 1}$$

$$r = \frac{1}{stride^{degree}}$$

The above sum is the additional percent of file, as multiple of D, to compute the size of the additional multi-resolution files. r in the above equation is a substitution variable to simplify the size of a resolution level k. This sum of r^k for all k will not exceed 1, assuming reasonable bounds ($stride > 1$, $height > 1$, $degree > 0$ and *stride*, *height* and *degree* are all integers). Therefore the total size required to store the original file and the sum of multi-resolution files will not ever exceed 2 times the original data size, and likely to be less in most cases.

Choosing *degree* is usually selected based upon the dimensionality of the data. If the user has 2D data, then a *degree* of 2 should be used. Likewise for 3D data, a *degree* of 3 should be used. In these cases, the data will be reduced in each dimension when generating resolution levels. In our study of POP ocean data, we used a *degree* of 2, even though it is 3D data, since the data is relatively "flat" in the z component.

Currently, choosing a good *stride* and *height* is more difficult. They dictate the relative quality in image change from one resolution to the next and number of resolution levels. Specifying *stride* depends on user preference on how much time they are willing to wait between resolution level updates and if the change in image was adequate enough for the time taken. Given that the data size of a resolution k is $D \times r^k$, and given a bandwidth b, we can solve for the time taken to transfer a data set, which is $D \times r^k / b$, assuming that server processing time is 0.

height determines the the lowest resolution level data fraction, $r^{height-1}$, and the number of levels between highest and lowest resolution. A user study would be necessary to derive a user satisfaction curve based on *height* and *stride*, measured across time taken, image quality change, lowest resolution, and number of resolution changes. Though ultimately, we suspect that good *stride* and *height* for user satisfaction is heavily user and data driven. In our observation, a *height* of 5 and a *stride* of 2 were satisfactory for the POP data. It may be likely that these values are good starting points.

Though there is an upper limit on *height*. As *height* controls the number of resolution levels, it also determines the lowest resolution data and data piece size. Given *height*, the lowest resolution data is size $D \times r^{height-1}$. Streaming piece sizes are also the same size as the lowest resolution. Therefore, the highest resolution data is broken into $stride^{degree^{height-1}}$ pieces. If *height* is too large then the streaming piece size becomes very small, and network and processing latency becomes a dominating factor in streaming the data. This, in particular, is why image-based distance visualization can have poor interactivity and frame rates. The time to transfer a data set at the highest resolution, assuming that there is one network send per piece, where the latency is l and the band-

width is b, is $l \times stride^{degree^{height-1}} + D/b$. It is clear that $l \times stride^{degree^{height-1}}$ can become an overriding time factor, and *height* should not be too large. This can be circumvented by sending multiple pieces at once, by grouping piece requests into one request and network send, to reduce the number of sends and network latency, assuming the client can handle multiple pieces.

In our implementation, we do not apply box filtering, gaussian filtering, wavelet decomposition, or or space filling curve reorganization methods to generate our multi-resolution data. While these methods can be easily used within our framework to generate multi-resolution levels, they can significantly increase pre-processing times due to nearly random read and/or write access to the disk and potentially multiple passes. For comparison, we use a regular striding to scan through the file exactly once in linear order and dump all resolution levels in one pass. This allows us to generate multi-resolution levels in approximately the same time it takes to read the data set once. The drawback to this method is that strided sampling will introduce aliasing error greater than than using a box filter, gaussian filter, or wavelet method. We make this cost/quality trade-off in order to have a very fast pre-processing step. It is important to note the "correct" data will eventually be shown in our visualization system at the highest resolution in time.

The additional benefit of reading the data in and constructing the multi-resolution data is that we are also able to collect the meta-data information, as well, which is used for prioritization and culling. This additionally speeds up the visualization pipeline by ignoring data that can be culled, and sending important high resolution data first.

4.2 Multi-resolution Renderer

The multi-resolution streaming renderer runs on the client side and primarily drives the piece processing requests downstream to the reader. The multi-resolution renderer operates similarly to the streaming VTK pipeline renderer, but requests pieces that refine high priority, low resolution pieces that will increase the resolution.

To render a visualization, given a particular view frustum, the renderer can determine what data should be requested by the priority queue and a kd-tree type splitting algorithm. Data outside of the view frustum is "wasted", and therefore not requested in this system. If a piece is on top of the priority queue, it is split spatially, and pieces of the next higher resolution are requested. Splits are made along the longest axis. A parent piece may be split into 2^{degree} pieces, and the request for those pieces are sent down the pipeline. The request is met by the reader, which sends the pieces back up the pipeline to be filtered and sent to the client for rendering.

The benefit of this process is that there is an immediate image displayed on the client by using low-resolution data that the network can quickly send and then progressively update the visualization over time. This is roughly calculated by $timeToImage = latency + dataSize/networkBandwidth$.

The rendering logic for the multi-resolution renderer follows:

- request and render the lowest resolution data
- put the lowest resolution data in a priority queue
- while priority queue is not empty
  - dequeue top element
  - split it into children
  - calculate the priority of the children, or determine if they can be culled
  - request the remaining children in priority order, and render them
  - push the children into the priority queue

The renderer will continue until there can be no more refinement, that is, full-resolution data has been retrieved. This can continuously happen while visual interaction takes place, which is one of the benefits of client side rendering. On zoomed in areas, the renderer will be able to finish quickly since many pieces will fall outside the view frustum.

Piece caching is a client side optimization, for fast rerendering, assuming that the client has memory to spare on top of piece size. There are various cache replacement policies that can used such as least recently used (LRU), in the case that you expect the user to zoom in and pan a lot. If you expect the user to zoom in, and then zoom out, a more logical replacement policy should be that lowest resolution pieces are held the longest with an LRU policy on top of that. Higher resolution pieces will always be flushed before low resolution pieces are replaced in that case.

4.3 Pipeline Information Exchange

With our multi-resolution reader and renderer in a new VTK/ParaView pipeline, we additionally have to modify the meta-information data structure that is passed in the visualization pipeline. In the pipeline, meta-information is passed upstream and downstream for notification of data modification for upstream sinks and downstream data requests. In the previous streaming architecture [1], the meta-data is attached for prioritization and culling through a PRIORITY meta-data key, along with data characteristics, such as min and max values. These previous pipeline modifications allows the client side renderer to sort and prioritize data chunk requests so that more important pieces can be streamed in first.

For our system, there are additional modifications to this meta-data that are necessary for the multi-resolution rendering. These additions allow the reader and renderer to communicate requested pieces at various resolution levels and spatial extents. In our implementation, we introduce the REQUEST_RESOLUTION key and utilize the existing REQUEST_UPDATE_EXTENT keys. When updating the image frame, the renderer will request pieces of various spatial ranges and resolution levels through these meta-data keys. During the information and execution phase of the reader, it will map the REQUEST_UPDATE_EXTENT and REQUEST_RESOLUTION meta-data into the appropriate data piece which will be sent back upstream to the renderer.

Filters do not need observe the multi-resolution keys in the pipeline and act upon them. Thus, most filters, in particular algorithms that are only local modifications or embarrassingly parallel, do not need to be aware of the new multi-resolution keys and will work as-is. Examples of filters that do work unmodified can be seen in Figure 11. An example filter that will not work as-is is the streamline filter, since it is a global algorithm. A streamline filter would need to be specifically be updated for to work in this multi-resolution system, similar to a parallel streamlines algorithm.

For prioritization, it is the case that many filters, if not all filters, change the data in the pipeline. Therefore, filters may invalidate any meta-data associated with a piece, such as min-max values or position. Since prioritization and culling is based on meta-data, a filter has to update any meta-data that is changed through modification. This can be made note of by a filter through additional flags introduced into the meta-data structure. If a filter does not update meta-data, in our implementation, prioritization mechanism conservatively assumes the worst, and prioritization will be disabled by default because the meta-data is unreliable.

5. RESULTS

To ground our research work with a real-world application problem, we are working closely with climate simulation scientists at Los Alamos National Laboratory (LANL). The LANL climate team uses the Jaguar supercomputer at Oak Ridge National Laboratory (ORNL) to run their Parallel Ocean Program (POP) ocean simulations. The team is generating approximately 24 fields, on a grid of 3600 by 2400 by 42 (33 GBs total), at 40 time steps a day. The generated data size is approximately 1.5 terabytes a day. At a measured transfer rate of 10 Mbps between LANL and ORNL (which actually is more realistically less than half of that due to firewalls), this would take approximately 370 hours (over 15 days) to transfer the entire run from ORNL back to LANL. Due to the size of the data and the remote location, a scalable, remote visualization solution was needed.

The tests for our system were performed with a modified version of ParaView that implements multi-resolution streaming. This was run on a Mac Pro 2 × 3GHz Quad Core Xeon with 16 GB of memory with a Seagate SATA ST3250820AS P disk drive (manufacturer reports 300MB/s peak disk bandwidth and 8 MB disk cache). Both the client and server were run on the computer, and we varied the tests between 1Mb to 1Gb simulated bandwidth with a 100ms latency. Before every test the operating system file cache was flushed, otherwise the data set would be cached in memory skewing the timings.

The following timings were done with a single field of float data from the POP data set. This data set at full resolution is $3600 \times 2400 \times 42$ at 1.35 GB. For the multi-resolution data, we used $stride = 2$, $height = 5$ and $degree = 2$. We refer to specific resolutions of the data as $n \times n \times 1$ to represent the sampling of the full-resolution data in x and y by n and z by 1. The splitting on refinement occurs in the x and y dimensions for each level since they are much longer than the z dimension. The original data is virtually broken into 256 spatial pieces. The resolutions are: $1800 \times 1200 \times 42$ in 64 virtual pieces of 346 MB total, $900 \times 600 \times 42$ in 16 virtual pieces of 86.5 GB total, $450 \times 300 \times 42$ in 4 virtual pieces of 22.6 MB total, and the lowest resolution is $225 \times 150 \times 42$ in 1 piece at 5.4 MB. The multi-resolution data representation on disk only takes $1/3 * 1.35GB$ extra disk space.

In the following test, we performed a surface rendering of the POP ocean data of the whole data set, with an estimate of 8Mbps network speed and 100ms latency. The major win for our system is an immediate good quality image, with increasing refinement over time, seen in Figure 8. Our method jumps to nearly 90% image accuracy (measured by CIELUV color space distance from the final image, normalized by maximum error) in the first frame, while the standard streaming method slowly creeps up to a good quality image.

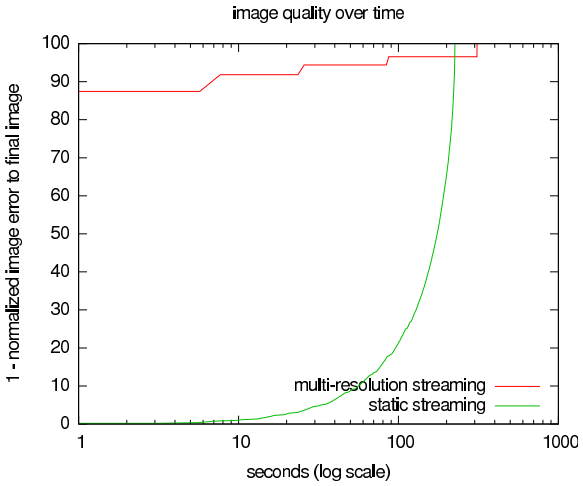

Figure 8: Comparison of standard VTK streaming (as in Figure 5) vs. our new multi-resolution streaming (as in Figure 6) of image quality over time. In our new system, we immediately get an image that is of decent quality, and improve it from that point on. The times are based on an estimate of 8Mbps network and 100ms latency.

Standard streaming, starts with a blank image incrementally completing portions of the image frame over time. Multi-resolution streaming sends a low resolution representation, shown almost instantaneously and then begins to stream pieces incrementally to augment the low-resolution data.

Full Extent	16x16x1	8x8x1	4x4x1	2x2x1	Full
Render	0.03 s	0.10 s	0.38 s	1.4 s	5.6 s

Table 3: Client side render times for rendering the whole spatial extent of the POP data set at different resolutions.

Our multi-resolution method works for quick full overviews, as there are low delivery times for low resolution data, seen in the bottom curves of Figure 9. At our target bandwidth for our ORNL case of 10Mbps, it takes approximately 1 second to deliver and render the 16x16x1 low resolution data. After the representation is delivered, the user can interact with it continuously at a minimum (the render time is taken from the instance of the first incremental render) of 33 frames per second (.03 seconds per frame), seen from Table 3 showing renderings for the entire spatial extent. While the user is interacting with this data, higher resolution data is continuously being streamed to the client, improving the rendering as the data is acquired.

A caveat is that visualization for the full data set at the highest resolution takes a significant amount of time for the data, seen in the top curves of Figure 9. The times level off as bandwidth increases (to the right) because of constant costs, such as network latency, for the high resolution data. This is primarily due to the high cost of sending the data. This is solved through our prioritization and culling methods, seen in Figure 10, where we visualize a zoomed in portion of the

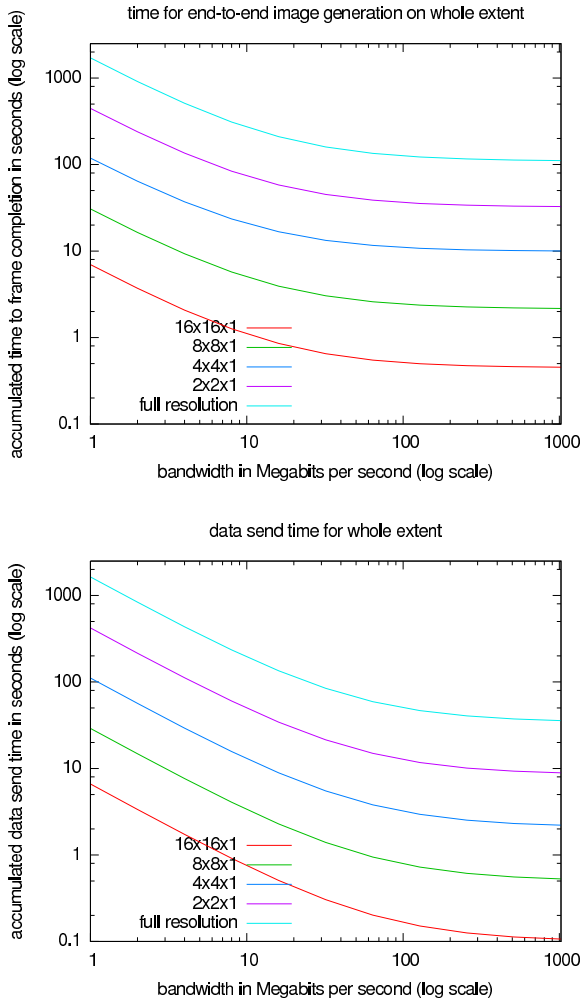

Figure 9: Times for end-to-end image completion (top graph) and data send (bottom graph) for various resolutions at various bandwidths for the whole spatial extent of the POP data set. Network latency is assumed to be 100 ms.

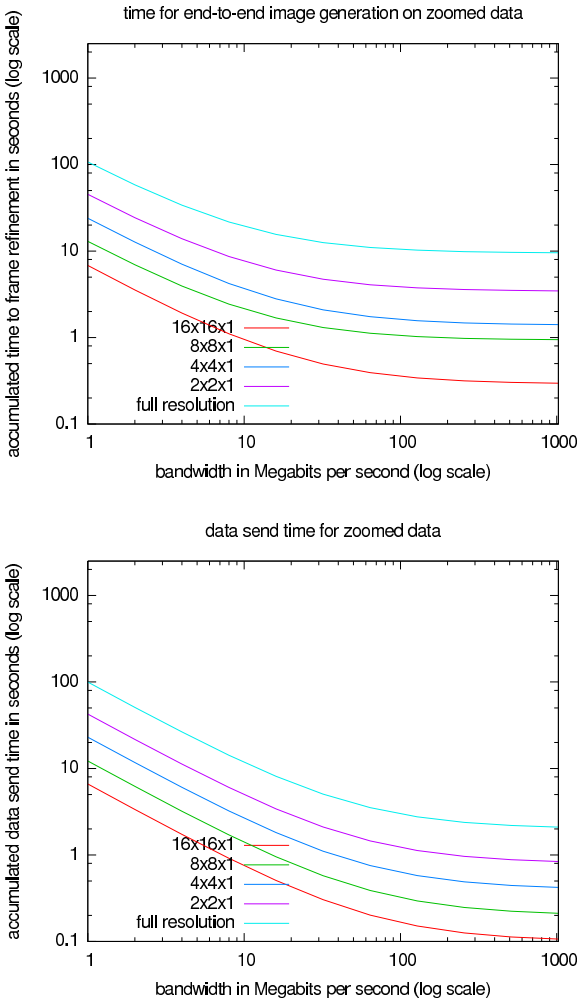

Figure 10: Times for end-to-end image completion and data send for various resolutions at various bandwidths for the zoomed in POP data set, seen in Figure 11. The network latency is assumed to be 100 ms.

data set.

Zoomed	16x16x1	8x8x1	4x4x1	2x2x1	Full
Render	0.03 s	0.03 s	0.05 s	0.09 s	0.27 s

Table 4: Client side render times for zoomed in portion of the POP data set seen in Figure 11.

The zoomed in area, shown in Figure 11, is a portion of the data set near the Antarctic Ocean. By prioritizing and only sending high resolution data for that view area, the time to end-to-end completion is significantly reduced, by an order of magnitude in time. This can be seen by comparing the graph times between Figure 9 and Figure 10. For example, with our 10Mbps scenario, the zoomed full resolution data for that area is delivered in 22 seconds. Before that point in time, the user is able to visualize low resolution versions of the data at high frame rate, seen in Table 4. The different times for delivery at can be seen in Figure 10, such that various resolutions are continuously delivered between 1 and 22 seconds.

Our disk scheme does not take a significant amount of time to create the multi-resolution representation on disk. In Table 5, we show the time to read the full data off of disk, which takes 30.0 seconds. For the system to create the additional multi-resolution data, the total time for our scenario was 46.5 seconds for 4 additional levels, which includes both the read and write times. If we go to the extreme case of creating 20 levels for the same data set, the time to write the multi-resolution data increases due to writing 20 files simultaneously, to approximately 5 times the read time. Though, 20 levels is impractical, because spatial chunks/read sizes become one float each, and sending one float at a time is not practical due to network latency.

Keeping the data as-is, but with additional multi-resolution files, has not had a significant impact on our read times for processing. Tables 6 and 7 show the read times for full extent multi-resolution streaming and zoomed extent multi-resolution streaming. The time to read the full data set, but in small spatial extents (streaming pieces), is

only 5.6 seconds longer at 35.6 seconds, rather than 30.0 seconds if a full linear read is done on the data. We can also get small spatial portions of the data quickly as can be seen in the read times for zoomed reading.

Time to Read Whole File	Time to Read and Create 4 levels	Time to Read and Create 20 levels
30.0 s	46.5 s	150.6 s

Table 5: Various timings for reading the original POP data and processing the POP data for multi-resolution levels.

Full Extent	16x16x1	8x8x1	4x4x1	2x2x1	Full
Read	0.18 s	0.92 s	5.0 s	11.8 s	35.6 s
Accum. Read	0.18 s	1.1 s	6.1 s	17.9 s	53.5 s

Table 6: Timings to read the whole spatial extent of the POP data set. Read times are the time to read the data a specific resolution level. Accum. Read times are the time to read the data a specific resolution level and all previous lower resolution levels.

Zoomed	16x16x1	8x8x1	4x4x1	2x2x1	Full
Read	0.18 s	0.47 s	1.4 s	2.8 s	6.1 s
Accum. Read	0.18 s	0.64 s	2.0 s	4.8 s	11.0 s

Table 7: Timings to read a zoomed spatial extent of the POP data set. The zoomed portion can be seen in Figure 11. Read times are the time to read the data a specific resolution level. Accum. Read times are the time to read the data a specific resolution level and all previous lower resolution levels.

6. CONCLUSION

We have described our prioritized multi-resolution streaming visualization system for distance visualization of large data. As shown by our results, we are able to provide overviews and prioritized details on demand quickly to the user. By the virtue of sending representational data, the client is able to perform fast rendering, to increase visual interactivity. This improvement to the VTK pipeline is general enough to work with existing visualization filters and analysis.

6.1 Acknowledgments

This work was funded by the US Department of Energy (DOE) Office of Science Advanced Scientific Computing Research (ASCR) program. It was also funded by the US Department of Energy (DOE) National Nuclear Security Administration (NNSA) Advanced Simulation & Computing (ASC) program. We would especially like to thank Kristi D. Brislawn of the Los Alamos National Laboratory and Erik W. Anderson of the University of Utah for their contributions to early development efforts. This work was primarily performed at the Los Alamos National Laboratory. Additionally, this research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

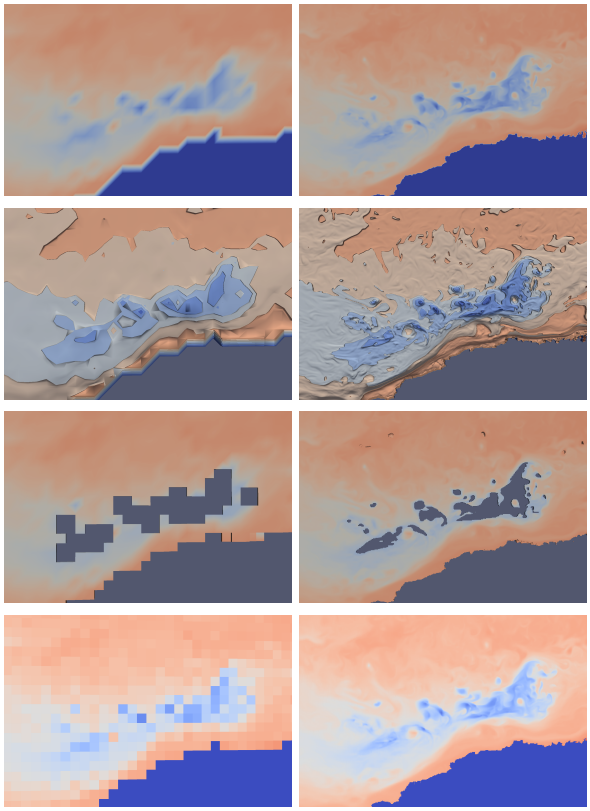

Figure 11: An example of our multi-resolution system visualizing a zoomed in portion of the POP data set with arbitrary filters and representations in VTK/ParaView. The left hand images are coarsest resolution, and the right hand images are the highest resolution. Top to bottom are surface rendering, isosurfacing, thresholding, and point rendering.

7. REFERENCES

[1] J. Ahrens, N. Desai, P. McCormick, K. Martin, and J. Woodring. A modular, extensible visualization system architecture for culled, prioritized data streaming. In *Visualization and Data Analysis 2007*, volume 6495, pages 64950I–1 – 64950I–12. SPIE, 2007.

[2] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. In *The Visualization Handbook*, pages 717–731. Academic Press, 2005.

[3] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote large data visualization in the paraview framework. In *6th Eurographics Symposium on Parallel Graphics and Visualization*, pages 163–170, May 2006.

[4] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A contract based system for large data visualization. *IEEE Visualization, 2005. VIS 05*, pages 191–198, 2005.

[5] H. Childs, M. Duchaineau, and K. Ma. A scalable, hybrid scheme for volume rendering massive data sets. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization, Lisbon, Portugal*, 2006.

[6] J. Clyne and M. Rast. A prototype discovery

environment for analyzing and visualizing terascale turbulent fluid flow simulations. In *Visualization and Data Analysis 2005*, pages 284–294. SPIE, January 2005.

[7] W. T. Corrêa, J. T. Klosowski, and C. T. Silva. Visibility-based prefetching for interactive out-of-core rendering. In *IEEE Symp. on Parallel and Large-Data Visualization and Graphics*, pages 1–8, 2003.

[8] J. T. Klosowski and C. T. Silva. Rendering on a budget: a framework for time-critical rendering. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 115–122, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[9] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 355–361, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[10] C. C. Law, A. Henderson, and J. P. Ahrens. An application architecture for large data visualization: A case study. In *Parallel and Large-Data Visualization and Graphics Symposium*, 2001.

[11] E. J. Luke and C. D. Hansen. Semotus visum: a flexible remote visualization framework. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 61–68, Washington, DC, USA, 2002. IEEE Computer Society.

[12] A. Norton and A. Rockwood. Enabling view-dependent progressive volume visualization on the grid. *IEEE Computer Graphics and Applications*, 23(2):22–31, 2003.

[13] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 2–2, New York, NY, USA, 2001. ACM.

[14] S. Prohaska, A. Hutanu, R. Kahler, and H.-C. Hege. Interactive exploration of large remote micro-ct scans. In *IEEE Visualization '04*, pages 345–352, Oct. 2004.

[15] L. P. Roy Whitney. DOE science networking challenge: Roadmap to 2008. Technical report, U.S. Department of Energy, Office of Science, ASCR, 2003.

[16] S. Rusinkiewicz and M. Levoy. Streaming qsplat: a viewer for networked visualization of large, dense models. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 63–68, New York, NY, USA, 2001. ACM.

[17] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics*. Kitware, 2003.

[18] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):176–187, 2003.

[19] C. Wang, J. Gao, L. Li, and H.-W. Shen. A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Volume Graphics, 2005. Fourth International Workshop on*, pages 11–223, June 2005.